|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-3 «Управление и моделирование систем»

**ОТЧЕТ   
о выполнении лабораторной работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Программная реализация нелинейных структур»**

**Вариант № \_\_\_**

Выполнил: студент 2 курса

группы \_\_\_\_\_\_\_\_\_\_

шифр \_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
*(фио студента)*

Проверил:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Москва 2020 г.

**Задание на лабораторную работу № 1.**

В рамках лабораторной работы №1 требуется программно реализовать (с помощью указателей (однонаправленных/двунаправленный динамический линенйый связанный список, массива или используя стандартный контейнер библиотеки STL “stack” или «queue» - по варианту) абстрактный тип данных (АТД) в соответствии с заданием (стек, дек, очередь с одной головой, очередь с головой и хвостом).

Абстрактный тип данных должен позволять осуществлять только операции, присущие типу линейного связанного списка:

* получить значение первого элемента (на выходе),
* добавить элемент (на вход),
* удалить элемент из списка (на выходе),
* проверить – список пуст,
* обнулить (проинициализировать) список (конструктур, при необходимости).
* деструктор (при необходимости)

Используя разработанный АТД и указанный набор операций, необходимо реализовать заданный алгоритм сортировки последовательности элементов заданного типа, при этом следует учитывать, что разрешен доступ (чтение/извлечение) только к элементу на выходе.

На основе исходного текста программы получить аналитическую оценку трудоемкости работы алгоритма сортировки, используя О-символику для каждого реализованного метода АТД и сортировки в целом.

**Вариант № \_\_\_.**

**Реализация связи элементов линейного списка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Способ организации линейного связанный список: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Алгоритм сортировки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Теория о сортировках.**

**Сортировка естественным двухпутевым слиянием.**

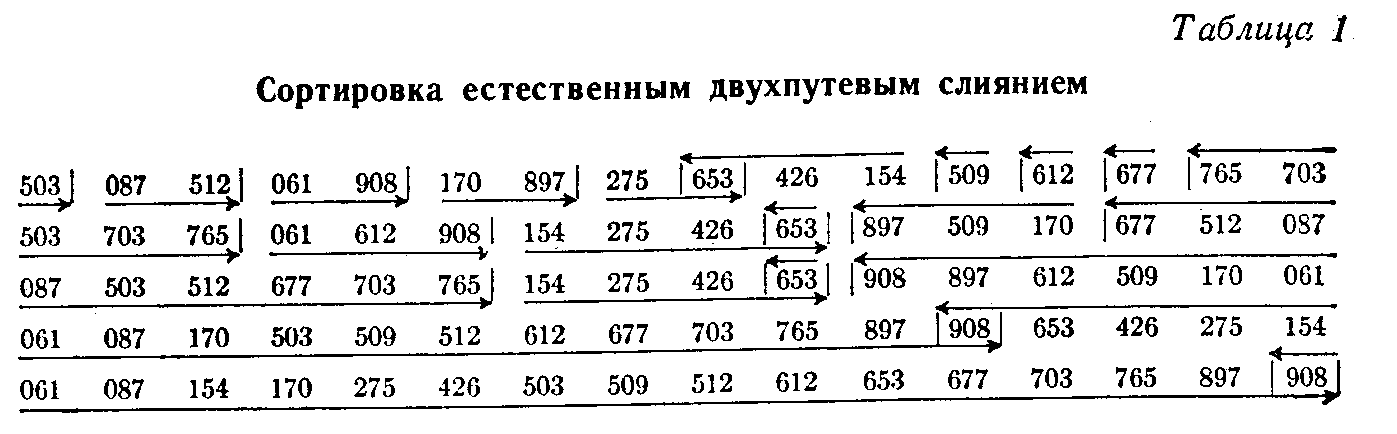
****Метод сортировки вставками – это есть слияние двух подфайлов, у одного из которого =1! Следовательно, задачу сортировки можно свести к слияниям, сливая все более длинные подфайлы до тех пор, пока не будет отсортирован весь файл. Для ускорения процесса вставок, можно рассмотреть вставку нескольких элементов за раз (n>1), группируя их. Такой метод слияний – один из самых первых методов сортировки, предложенный в 1945 Джоном фон Нейманом и носит название естественное двухпутевое слияние.

Рисунок иллюстрирует сортировку слиянием, когда мы продвигаемся с обоих концов, подобно методам быстрой сортировки, обменной сортировки и т.д. Мы анализируем исходный файл слева и справа, двигаясь к середине. Пропустим первую строку и рассмотрим вторую. Слева мы видим возрастающий отрезок 503 703 765, а справа, если читать справа налево, имеем отрезок 087 512 677. Слияние этих двух последовательностей дает подфайл 087 503 512 677 703 765, который перемещается слева в строке 3. Затем ключи 061 612 908 в строке 2 сливаются с 170 509 987 и результат записывается справа в строке 3. Наконец, 154 275 426 653 сливается с 653 (перекрытие обнаруживается раньше, чем оно может привести к нежелательным последствиям) и результат записывается слева. Точно также строка 2 получилась из исходного файла в строке 1.

Вертикальными линиями на рисунке отмечены границы между отрезками. Это так называемые ступеньки вниз, где меньший элемент следует за большим. В середине файла возникает двусмысленная ситуация, когда мы с обоих концов читаем один и тот же ключ.

Описанный метод называют "естественным" слиянием, потому что он использует отрезки, которые "естественно" образуются в файле.

**Листинг программы с расчетами.**

#include "stdafx.h"

#include <locale.h>

#include <ctime>

#include "windows.h"

#include <iostream>

using namespace std;

struct Node

{

// указатель на следующий элемент в очереди

Node \*next;

// значение элемента в очереди

int value;

};

class Ochered

{

public:

// Колличество элементов в списке

int N;

int i;

int size = 0;

int size\_1 = NULL;

// Счетчик колличества операций

unsigned long long int N\_op = 0;

// указатель на первый элемент

Node \*head;

// указатель на хвост очереди

Node \*tail;

// инициализация очереди

void Init() //2

{

head = NULL; //1

// При инициализации head и tail = NULL

tail = head; //1

}

// добавление элемента в начало очереди, x - значение этого элемента

void Add(int x) //(3+2+2)+(1+2+1)+2=13

{

// создание нового элемента

Node \*node = new Node; //3

// в поле next данного элемента записывается NULL

node->next = NULL; //2

// в поле value записывается значение ячейки очереди int

node->value = x; //2

// после инициализации и при создании первого элемента мы проходим по ветке else

if (tail != NULL) //1

{

/\* добавление элемента через хвост в очереди,

tail указывает на первый элемент в очереди перед добавляемым

в поле next первого элемента записывается адрес нового первого элемента\*/

tail->next = node; //2

// tail начинает указывать на новый элемент в очереди

tail = node; //1

}

else

{

// хвост указывает на первый элемент в очереди

tail = node; //1

// голова указывает на первый элемент в очереди

head = tail; //1

}

size++; //2

}

// проверка очереди на пустоту

bool Isempty() //1

{

//true, если head = NULL; false, если head указывает на какой-либо элемент в очереди

return head == NULL; //1

}

// удаление элемента из конца очереди

int Del() //1+(2+2+2+1+2+2+2)=14

{

int val = NULL; //1

// если очередь не пуста

if (Isempty() != 1) //2

{

// создаем указатель на структуру

Node \*tmp;

// в указателюь tmp кладем ссылку на next предпоследнего элемента в очереди

tmp = head->next; //2

// в val записываем значение последнего элемента в очереди

val = head->value; //2

// удаляем объект по указателю head (последний элемент в очереди)

delete head;

// head начинает указывать на предпоcледний элемент в очереди (последний удален)

head = tmp; //1

// После удаления последнего элемента из очереди происходит повторная инициализация

if (Isempty() == 1) //2

{

Init(); //2

}

size--; //2

// возвращаем значение последнего элемента в очереди

return val;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

// получение значения из конца очереди

int Value() //3

{

// если очередь не пуста

if (Isempty() != 1) //2

{

// через указатель head получить значение value последнего элемента в очереди

return head->value; //1

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

};

// Класс наследник

class Numbers : public Ochered

{

public:

// получение значения из списка, x - номер элемента в списке

int get(int x) //2+1+1+(1+33n)=5+33n

{

// если очередь не пуста

if (Isempty() != 1) //2

{

size\_1 = size; //1

int val1 = NULL; //1

// Перебор всех значений в списке

for (int i = 1; i < size\_1 + 1; i++) //

{

if (i == x) //1

{

// Когда i доходит до номера получаемого элемента, то в val записывается значение данного элемента

val1 = Value(); //4

}

// Перебор начинается только в случае если x не равен нулю

if (x != 1) //1

{

// Вначале элемент удаляется из конца очереди, и сразу же добавляется в начало очереди, и так в цикле пока не восстановится первоначальное состояние очереди

Add(Del()); //14+13=27

}

else

{

// В противном случае сразу выходим из цикла for

break;

}

}

// Возвращается значение элемента под номером x из очереди

return val1;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

// установка значения элемента в списке, x - номер элемента в списке, y - значение элемента в списке

void set(int x, int y) //2+3+31n=5+31n

{

// если очередь не пуста

if (Isempty() != 1) //2

{

if (x <= size) //1+1+1+31n=3+31n

{

size\_1 = size; //1

for (int i = 1; i < size\_1 + 1; i++) //

{

if (i == x) //1

{

// Запись значения y в список

head->value = y; //2

}

if (x != 1) //1

{

Add(Del()); //14+13=27

}

else

{

break;

}

}

}

// Если элемент устанавливается за пределы изначального списка

else //

{

if (x == size + 1) //2

{

//Если номер элемента в списке на 1 больше размера очереди, значит элемент добавляется в начало очереди

Add(y);

}

else

{

for (i = size + 1; i < x; i++) //

{

// В противном случае очередь заполняется нулями

Add(0); //13

}

Add(y); //13

}

}

}

else { cout << "Очередь пуста!" << endl; }

}

void sort(int N)

{

int s, f, d, i, j, g, k, p;

s = 1; //1

do

{

s = 1 - s; //2

d = 1; //1

f = 1; //1

if (s == 0) //1

{

i = 1; //1

j = N; //1

k = N + 1; //2

g = 2 \* N; //2

}

else

{

i = N + 1; //2

j = 2 \* N; //2

k = 1; //1

g = N; //1

}

while (i != j)//

{

// if(K[i]<K[j])

if (get(i) > get(j))//

{

// R[k] = R[j];

set(k, get(j));//

k = k + d; //2

j = j - 1; //2

// if (!(K[j+1]<=K[j]))

if (get(j + 1) <= get(j)) //5+66n+1+5+66n+1=132n+12

{

}

else

{

do

{

// R[k]=R[i];

set(k, get(i)); //10+128n

k = k + d; //2

i = i + 1; //2

}

// while(!K[i-1]>=K[i]);

while (get(i - 1) < get(i)); //

f = 0; //1

d = -d; //1

p = k; //1

k = g; //1

g = p; //1

}

}

else

{

// R[k]=R[i];

set(k, get(i));

k = k + d; //2

i = i + 1; //2

// if(K[i-1]<=K[i])

if (get(i - 1) <= get(i)) //…+2

{

}

else

{

do

{

// R[k] = R[j];

set(k, get(j));

k = k + d; //2

j = j - 1; //2

}

// while(K[j+1]>=K[j]);

while (get(j + 1) < get(j)); //…+2

f = 0; //1

d = -d; //1

p = k; //1

k = g; //1

g = p; //1

}

}

}

//R[k]=R[i];

set(k, get(i)); //10+128n

} while (f != 1); //

if (s == 0) //1

{

for (int i = 1; i <= N; i++)//

{

// R[i] = R[i+n];

set(i, get(i + N)); //

}

}

size\_1 = size; //1

for (int i = 1; i <= size\_1; i++) //

{

if (i <= size / 2) //2

{

Add(Del()); //27

}

else

{

Del();

}

}

}

};

int main()

{

setlocale(LC\_ALL, "ru");

srand(time(NULL));

// Схема эксперимента

// Инициализация очереди и заполнение хранилища ключей

int i, t\_s, t\_f;

// Хранилище ключей

int Key[3000];

int N = 300;

Numbers list;

list.Init();

for (i = 0; i < 3000; i++)

{

// Заполнение хранилища ключей случайными числами до 1000

Key[i] = rand() % 999;

}

for (i = 0; i < 10; i++)

{

for (int z = N - 300; z < N; z++)

{

list.Add(Key[z]);

}

list.N\_op = 0;

t\_s = GetTickCount();

list.sort(N);

t\_f = GetTickCount();

cout << "Номер сортировки: " << i + 1 << " Колличество отсортированных элементов: " << N << " Время сортировки (ms): " << t\_f - t\_s << " Колличество операций (N\_op): " << list.N\_op << endl; // Шаг в 300 элементов

N = N + 300;

}

return 0;

}

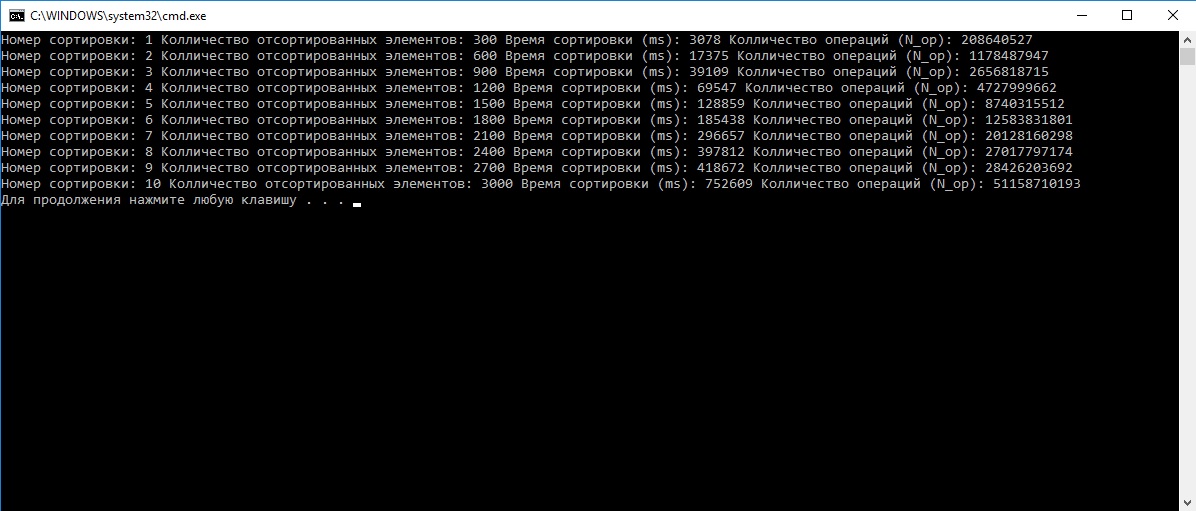
F(n)=

O(F(n))=

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 29195057069 | 222178104 | 3,078 | 208640527 |
| 600 | 260537859079 | 1993424837 | 17,375 | 1178487947 |
| 900 | 933376707073 | 7154246488 | 39,109 | 2656818715 |
| 1200 | 2303953779676 | 17675398697 | 69,547 | 4727999662 |
| 1500 | 4639045330098 | 35608770400 | 128,859 | 8740315512 |
| 1800 | 8213184364536 | 63065971907 | 185,438 | 12583831801 |
| 2100 | 13307060243479 | 102206003825 | 296,657 | 20128160298 |
| 2400 | 20206471287464 | 155227189577 | 397,812 | 27017797174 |
| 2700 | 29201584161663 | 224361472088 | 418,672 | 28426203692 |
| 3000 | 40586381821520 | 311870163205 | 752,609 | 51158710193 |

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 9485073,771 | 72182,619 | 139,929 | 1,064 |
| 14994984,695 | 114729,487 | 221,078 | 1,691 |
| 23866033,574 | 182930,949 | 351,313 | 2,692 |
| 33128010,980 | 254150,411 | 487,299 | 3,738 |
| 36000941,572 | 276339,024 | 530,764 | 4,074 |
| 44290729,864 | 340091,954 | 652,677 | 5,011 |
| 44856720,871 | 344525,845 | 661,116 | 5,077 |
| 50794021,516 | 390202,380 | 747,894 | 5,745 |
| 69748118,244 | 535888,409 | 1027,276 | 7,892 |
| 53927579,688 | 414385,375 | 793,342 | 6,096 |

**Скриншот работы программы:**

**Выводы.**

По результатам экспериментов было установлено, что графики C1, C2, C3, C4 от N имеют линейную зависимость от количества элементов.

**Литература:**

1. Структуры данных и алгоритмы. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. – М.: Издательский дом «Вильямс», 2000

2. Д. Кнут. Искусство программирования для ЭВМ.

**Приложение 1. Применение счетчика операций N\_op.**

#include "stdafx.h"

#include <locale.h>

#include <ctime>

#include "windows.h"

#include <iostream>

using namespace std;

struct Node

{

// указатель на следующий элемент в очереди

Node \*next;

// значение элемента в очереди

int value;

};

class Ochered

{

public:

// Колличество элементов в списке

int N;

int i;

int size = 0;

int size\_1 = NULL;

// Счетчик колличества операций

unsigned long long int N\_op = 0;

// указатель на первый элемент

Node \*head;

// указатель на хвост очереди

Node \*tail;

// инициализация очереди

void Init()

{

head = NULL; N\_op++;

// При инициализации head и tail = NULL

tail = head; N\_op++;

}

// добавление элемента в начало очереди, x - значение этого элемента

void Add(int x)

{

// создание нового элемента

Node \*node = new Node; N\_op+=3;

// в поле next данного элемента записывается NULL

node->next = NULL; N\_op+=2;

// в поле value записывается значение ячейки очереди int

node->value = x; N\_op+=2;

// после инициализации и при создании первого элемента мы проходим по ветке else

if (tail != NULL)

{ N\_op++;

/\* добавление элемента через хвост в очереди,

tail указывает на первый элемент в очереди перед добавляемым

в поле next первого элемента записывается адрес нового первого элемента\*/

tail->next = node; N\_op+=2;

// tail начинает указывать на новый элемент в очереди

tail = node; N\_op++;

}

else

{ N\_op++;

// хвост указывает на первый элемент в очереди

tail = node; N\_op++;

// голова указывает на первый элемент в очереди

head = tail; N\_op++;

}

size++; N\_op+=2;

}

// проверка очереди на пустоту

bool Isempty()

{ N\_op++;

//true, если head = NULL; false, если head указывает на какой-либо элемент в очереди

return head == NULL;

}

// удаление элемента из конца очереди

int Del()

{

int val = NULL; N\_op++;

// если очередь не пуста

if (Isempty() != 1)

{ N\_op++;

// создаем указатель на структуру

Node \*tmp;

// в указателюь tmp кладем ссылку на next предпоследнего элемента в очереди

tmp = head->next; N\_op+=2;

// в val записываем значение последнего элемента в очереди

val = head->value; N\_op+=2;

// удаляем объект по указателю head (последний элемент в очереди)

delete head;

// head начинает указывать на предпоcледний элемент в очереди (последний удален)

head = tmp; N\_op++;

// После удаления последнего элемента из очереди происходит повторная инициализация

if (Isempty() == 1)

{ N\_op++;

Init();

}

size--; N\_op+=2;

// возвращаем значение последнего элемента в очереди

return val;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

// получение значения из конца очереди

int Value()

{

// если очередь не пуста

if (Isempty() != 1)

{ N\_op+=2;

// через указатель head получить значение value последнего элемента в очереди

return head->value;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

};

// Класс наследник

class Numbers : public Ochered

{

public:

// получение значения из списка, x - номер элемента в списке

int get(int x)

{

// если очередь не пуста

if (Isempty() != 1)

{ N\_op++;

size\_1 = size; N\_op++;

int val1 = NULL; N\_op++;

/\* Перебор всех значений в списке\*/ N\_op++;

for (int i = 1; i < size\_1 + 1; i++)

{

if (i == x)

{ N\_op++;

// Когда i доходит до номера получаемого элемента, то в val записывается значение данного элемента

val1 = Value(); N\_op++;

}

// Перебор начинается только в случае если x не равен нулю

if (x != 1)

{ N\_op++;

// Вначале элемент удаляется из конца очереди, и сразу же добавляется в начало очереди, и так в цикле пока не восстановится первоначальное состояние очереди

Add(Del());

}

else

{ N\_op++;

// В противном случае сразу выходим из цикла for

break;

}

}

// Возвращается значение элемента под номером x из очереди

return val1;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

// установка значения элемента в списке, x - номер элемента в списке, y - значение элемента в списке

void set(int x, int y)

{

// если очередь не пуста

if (Isempty() != 1)

{ N\_op++;

if (x <= size)

{ N\_op++;

size\_1 = size; N\_op++;

N\_op++;

for (int i = 1; i < size\_1 + 1; i++)

{

if (i == x)

{ N\_op++;

// Запись значения y в список

head->value = y; N\_op+=2;

}

if (x != 1)

{ N\_op++;

Add(Del());

}

else

{ N\_op++;

break;

}

}

}

// Если элемент устанавливается за пределы изначального списка

else

{ N\_op++;

if (x == size + 1)

{ N\_op+=2;

//Если номер элемента в списке на 1 больше размера очереди, значит элемент добавляется в начало очереди

Add(y);

}

else

{ N\_op+=2; N\_op++;

for (i = size + 1; i < x; i++)

{

// В противном случае очередь заполняется нулями

Add(0);

}

Add(y);

}

}

}

else { cout << "Очередь пуста!" << endl; }

}

void sort(int N)

{

int s, f, d, i, j, g, k, p;

s = 1; N\_op++;

do

{

s = 1 - s; N\_op+=2;

d = 1; N\_op++;

f = 1; N\_op++;

if (s == 0)

{ N\_op++;

i = 1; N\_op++;

j = N; N\_op++;

k = N + 1; N\_op+=2;

g = 2 \* N; N\_op+=2;

}

else

{ N\_op++;

i = N + 1; N\_op+=2;

j = 2 \* N; N\_op+=2;

k = 1; N\_op++;

g = N; N\_op++;

}

while (i != j)//

{ N\_op++;

// if(K[i]<K[j])

if (get(i) > get(j))

{ N\_op++;

// R[k] = R[j];

set(k, get(j));

k = k + d; N\_op+=2;

j = j - 1; N\_op+=2;

// if (!(K[j+1]<=K[j]))

if (get(j + 1) <= get(j))

{

N\_op+=2;

}

else

{ N\_op+=2;

do

{

// R[k]=R[i];

set(k, get(i));

k = k + d; N\_op+=2;

i = i + 1; N\_op+=2; N\_op+=2;

}

// while(!K[i-1]>=K[i]);

while (get(i - 1) < get(i));

f = 0; N\_op++;

d = -d; N\_op++;

p = k; N\_op++;

k = g; N\_op++;

g = p; N\_op++;

}

}

else

{ N\_op++;

// R[k]=R[i];

set(k, get(i));

k = k + d; N\_op+=2;

i = i + 1; N\_op+=2;

// if(K[i-1]<=K[i])

if (get(i - 1) <= get(i))

{

N\_op+=2;

}

else

{ N\_op+=2;

do

{

// R[k] = R[j];

set(k, get(j));

k = k + d; N\_op+=2;

j = j - 1; N\_op+=2; N\_op+=2;

}

// while(K[j+1]>=K[j]);

while (get(j + 1) < get(j));

f = 0; N\_op++;

d = -d; N\_op++;

p = k; N\_op++;

k = g; N\_op++;

g = p; N\_op++;

}

}

}

//R[k]=R[i];

set(k, get(i)); N\_op++;

} while (f != 1);

if (s == 0)

{ N\_op+=2;

for (int i = 1; i <= N; i++)

{ N\_op++;

// R[i] = R[i+n];

set(i, get(i + N));

}

}

size\_1 = size; N\_op+=2;

for (int i = 1; i <= size\_1; i++)

{

if (i <= size / 2)

{ N\_op+=2;

Add(Del());

}

else

{ N\_op+=2;

Del();

}

}

}

};

int main()

{

setlocale(LC\_ALL, "ru");

srand(time(NULL));

// Схема эксперимента

// Инициализация очереди и заполнение хранилища ключей

int i, t\_s, t\_f;

// Хранилище ключей

int Key[3000];

int N = 300;

Numbers list;

list.Init();

for (i = 0; i < 3000; i++)

{

// Заполнение хранилища ключей случайными числами до 1000

Key[i] = rand() % 999;

}

for (i = 0; i < 10; i++)

{

for (int z = N - 300; z < N; z++)

{

list.Add(Key[z]);

}

list.N\_op = 0;

t\_s = GetTickCount();

list.sort(N);

t\_f = GetTickCount();

cout << "Номер сортировки: " << i + 1 << " Колличество отсортированных элементов: " << N << " Время сортировки (ms): " << t\_f - t\_s << " Колличество операций (N\_op): " << list.N\_op << endl;

// Шаг в 300 элементов

N = N + 300;

}

return 0;

}